FEWZ 2.1: A User's Guide

Frank Petriello, Seth Quackenbush,
Ryan Gavin, Ye Li

last update: February 29, 2012

Contents

Introduction

FEWZ is a Fortran based, numerical code that calculates electroweak (EW) gauge boson production at hadron colliders through $\mathcal{O}(\alpha_s^2)$ in perturbative QCD. The leptonic decays of the Z or W with full spin correlations are included, as well as finite width effects, and, in the case of Z production, the Z/γ^* interference. The calculation is fully exclusive, allowing the user to investigate the production cross sections as they wish by placing phase space constraints on the EW gauge boson, final state leptons, or jets. FEWZ also has the ability to create predefined histograms simultaneously, where the parameters of the histograms can be set by the user. All numerical results presented include PDF uncertainties.

Numerical integration within FEWZ is performed using Vegas, which is part of the Cuba library. Cuba-1.7 is distributed with FEWZ. Basic system requirements include Fortran and C++ compilers, Python, and the bash shell.

This guide describes the running of FEWZ. Articles discussing the physics of this code can be found here: http://arxiv.org/abs/hep-ph/0609070, http://arxiv.org/abs/1011.3540, and http://arxiv.org/abs/1201.5896.

Getting Started

FEWZ is distributed as a compressed tar file. To untar, use the command

```
> tar zxvf FEWZ_2.1.tar.gz
```
or
```
> tar xvf FEWZ_2.1.tar
```
(if unzipped).

This will create the FEWZ directory. It should contain seven subdirectories and the makefile. The following is a brief introduction to the directory structure.

Cuba-1.7: contains the Cuba library for numerical integration.

dat: contains all PDF files.

include: contains files to be included in various subroutines and functions found in the `src` directory.

mainSrc: contains the main `FEWZ` source files.

src: directory containing the functions which construct the differential cross section, perform histogramming, implement cuts, etc.

pdfSrc: directory containing PDF collaborations' source code for calling their PDFs.

bin: contains executables and input files for running `FEWZ`, and the scripts for starting and analyzing those runs.

`FEWZ` takes advantage of the high-performance computing system Condor, if it is present. By default the makefile attempts to generate the executable to run locally as well as the one to run on Condor. To compile and create the `FEWZ` executables, run

```
> make
```

`make` will first create the Cuba library (`libcuba.a`), and willl then continue to compile `FEWZ`. Sometimes there are problems in creating the Cuba library depending on the machine and compilers present. Such situations are documented on the Cuba website (http://www.feynarts.de/cuba/). It is recommended that if remedies are not readily found, that the user download the appropriate library file from the Cuba website, and place it within the `Cuba-1.7` directory. Once the Cuba library is present, running `make` again will proceed to compile `FEWZ`.

After compiling, `FEWZ` will create four executables and place them in the `bin` directory.

`fewzz` - executable for Z production.

`fewzw` - executable for W production.

`condor_fewzz` - Condor executable for Z production.

`condor_fewzw` - Condor executable for W production.

Note: If Condor is not present on the machine where the compiling is performed, an error will be output to screen. This occurs, however, after the executables to run locally have been created. You can avoid this error message by typing

```
> make fewzz
```

or

```
> make fewzw
```

The makefile will be discussed in more detail in a later section.

Inside the `bin` directory, one can now run `FEWZ`. The specifics of a run can be found and set within two types of files, the *input* file and the *histogram* file. There are different input files for Z and W production. In the histogram file, the user can specify the number of bins and the upper and lower bounds on the histogram. In this scenario, the bins will be equally spaced. However, accompanying the histogram file is another file which can be called to specify the exact boundaries of each bin to allow for bins of varying size. These four files appear in the `bin` directory as `input_z.txt`, `input_w.txt`, `histograms.txt`, and `bins.txt` by default. The user is free to rename these files.

Note: Individual lines within the input and histogram files cannot be added or deleted; they can only be altered. `FEWZ` expects a specifc structure in these files. This is not the case for the bin size defining file.

The input files (`input_z.txt`, `input_w.txt`), the histogram input file (`histograms.txt`), and the bin specifying file (`bins.txt`) will be discussed in more detail in a later section.

Local Run

Running the local version of `FEWZ` is controlled by the shell script **local_run.sh**. Its use is detailed below.

```
> ./local_run.sh  <boson>  <run_dir>  <usr_input_file>  <usr_histo_file>
              <output_file_extension>  <pdf_location>  <num_processors>
```

The various arguments have the following defintions.

`<boson>` is the mode that `FEWZ` is to be run in: `z` for Z production or `w` for W production. By specifying Z or W production, the `local_run.sh` script knows which executable, `fewzz` or `fewzw` respectively, to use.

`<run_dir>` is the name of the directory where all of the files for the job will be placed. It is created when the shell script is run.

<usr_input_file> is the user-defined input file.

<usr_histo_file> is the user-defined histogram input file.

<output_file_extension> is the name of the output files.

<pdf_location> is the location of the dat directory, the directory containing the PDF grids; if running from the bin directory, one can simply use .. for the default setup.

<num_processors> is the number of processes to be used (default = 1), and should be set to the number of processors wanted to work on the calculation.

Here is an example when running Z production from the bin directory:

```
> ./local_run.sh  z  my_run  my_input.txt  my_histograms.txt
              my_results.dat  ..  1
```

This will create a directory called my_run where the preliminary output files named my_results.dat will be placed. FEWZ will look for the files my_input.txt and my_histograms.txt in the bin directory.

The shell script **finish.sh** should be used to process the output files from FEWZ to produce a complete result from the run. The primary purpose of the finish.sh script is to calculate and present errors due to PDFs, as well as normalize specific histograms. This script has other uses, which will be described in a later section. finish.sh has the following command line structure.

```
> ./finish.sh  <run_dir>  <order_prefix>.<output_file_extension>
```

<run_dir> and <output_file_extension> are the user defined run directory and output file first declared in running local_run.sh, described above. <order_prefix> is the perturbative order of the run: LO for leading order, NLO for next-to-leading order, NNLO for next-to-next-to-leading order. This script will create one final output file in the bin directory, titled <order_prefix>.<output_file_extension>.

In keeping with the example above, if running at next-to-next-to-leading order (NNLO) in the bin directory, the finish.sh script would be used as

```
> ./finish.sh  my_run  NNLO.my_results.dat
```

This will create one final result file titled NNLO.my_results.dat.

Note: If running at orders LO or NLO, `finish.sh` can be used before the end of a run, i.e. before Vegas has reached the desired precision or maximum number of evaluations. As long as Vegas has performed one iteration and created output files, `finish.sh` will produce an output file. This is not true for NNLO. Here, the user must wait until all subdirectories contain output files before using `finish.sh`.

Note: We note that `finish.sh` is capable of performing binary operations on multiple runs. We detail these features in a later section.

Condor Run

The `condor_run.sh` script is designed to assist the user in setting up and running `FEWZ` on the Condor batch system. This script is executed just as `local_run.sh` is, with the exception of specifying the number of processors, i.e.

```
> ./condor_run.sh  <boson>  <run_dir>  <usr_input_file>  <usr_histo_file>
               <output_file_extension>  <pdf_location>
```

When processing Condor jobs, the `finish.sh` script can be used as it was described for the local run. As a reminder, once there exists output files in each of the subdirectories, `finish.sh` can be used. The jobs themselves might be far from finished, but running this script can give the user an idea of the status of the overall NNLO calculation.

Note: Due to the differences in user resources with respect to Condor, it is not unexpected that the Condor related scripts might have problems working "straight out of the box." Therefore, the Condor user might have to adjust the scripts to their Condor environment.

FEWZ Argument Files

Here the user will find detailed descriptions of the `FEWZ` argument files, `input_z.txt`, `input_w.txt`, and `histograms.txt` (as well as `bins.txt`).

It is important for the user to remember that the format of the argument files should not be changed. The internal program expects the format as is, and may crash or produce unexpected results if it receives something different.

Input File

The Z input file, `input_z.txt`, and the W input file, `input_w.txt`, only differ slightly. The files have the same structure, and any differences will be explicitly discussed.

Collider parameters - energies in GeV.

- collider CM energy.
- renormalization and factorization scales (μ_R and μ_F, respectively).
- collider type
  - Z **production** - pp (LHC) = 1, $p\bar{p}$ (Tevatron) = 2.
  - W **production** - pp (LHC) W^- = 1, $p\bar{p}$ (Tevatron) W^- = 2, pp (LHC) W^+ = 3, $p\bar{p}$ (Tevatron) W^+ = 4.

EW parameters - default input file sets parameters using the G_μ scheme, with G_F, M_Z, and M_W as the input parameters. The user has the freedom to separately set overall couplings, CKM matrix elements (W production only), and vector and axial couplings of quarks and leptons as desired. Definitions used for g_V^f and g_A^f (as in Z input file) follow.

- $g_V^f = 2(T_3^f)_L - 4Q^f \sin_w^2 \theta$
- $g_A^f = -2(T_3^f)_L$

Vegas parameters - parameters for numerical integration, including the random number seed.

- **desired precision** - It is recommended the user rely on the absolute accuracy. One can get an estimate of the cross section by first running at LO (or NLO when the lowest-order process begins at $\mathcal{O}(\alpha_s)$) for a few iterations, then set the desired absolute accuracy intelligently for the true run.
- **calls per iteration** - The initial number of samples. Due to the internal workings of Vegas, it is recommended that evaluations be set in increments of 1000.
- **increase calls per iteration** - It is recommended that this be nonzero and a multiple of 1000. Numerical convergence is usually improved by choosing a number such that the increase is on the order of the initial sample.

Perturbative order settings

- LO = 0, NLO = 1, or NNLO = 2
- Z **or** W **pole focus** - On =1, Off = 0. Pole focus *on* performs a variable transformation to undo the Breit-Wigner resonance, smoothing the integrand around the EW gauge boson mass pole. Off, the transformation is set to smooth the integrand using the user defined **Upper** and **Lower** invariant mass limits. This is more robust when sampling off the pole, but less efficient at numerical convergence on the pole.
- **turn photon off (**Z **production only)** - Yes = 1, No = 0. If Yes, the photon contribution is turned off, including the Z-photon interference.

Invariant mass cut - minimum and maximum invariant mass set in GeV.

Transverse mass cut - minimum and maximum transverse mass set in GeV.

Z **or** W **boson cuts** - rapidity, Y, and p_T in GeV.

Leptonic cuts -

Z production

- pseudorapidity, η, and p_T in GeV for both leptons.
- Minimum and maximum p_T cuts on leptons when ordered *by* p_T, i.e. *hard* and *soft* leptons.

W production

- pseudorapidity, η, and p_T in GeV for charged lepton.
- Minimum and maximum $p_{T,miss}$ (neutrino) in GeV.

- Minimum and maximum p_T cuts on charged lepton and missing energy (neutrino) when ordered *by* p_T, i.e. *hard* and *soft* leptons.

Jet cuts

- **Minimum pT for Observable Jet** - To properly define **Jet** observables, a minimum $p_{T_{jet}}$ must be set. The default is set to 20 GeV.
- **min, max # of Jets** - allows the user to choose the desired number of jets by setting the minimum and maximum number permitted. A final state parton is only labelled a jet if it survives the minimum pT for an observable jet (as set by the user). The check for number of jets is performed after the jet merging algorithm. The default is set to `min=0` and `max=2`, placing no constraints on the number of jets present.
- **Activate cuts on Jets 1, 2** - Flag allows user to choose if cuts involving jets should be implemented, where jets 1 & 2 are ordered by p_T. These cuts include jet p_T in GeV, pseudorapidity, η, and particle isolation.
- **Jet algorithm** - either k_T, *anti-*k_T, or *cone* algorithm can be used to merge final state partons into jets; the algorithm cone size, ΔR_{cone}, is set by the user. ΔR is defined as $\Delta R = \sqrt{\Delta\eta^2 + \Delta\phi^2}$.
- ΔR **separation for cone algorithm** - specific to the *cone* algorithm, this parameter places an upper limit on the parton-parton separation, by requiring that $\Delta R_{parton,parton} < \Delta R_{sep} \cdot \Delta R_{cone}$; default is $\Delta R_{sep} = 1.3$.

Particle isolation - isolation measure in ΔR; lepton-jet isolation check performed after jet merging.

Z production

- lepton – anti-lepton isolation
- lepton(anti-lepton) – jet isolation

W production

- charged lepton – jet isolation

Lepton – anti-lepton $\Delta\phi$ **cut** - minimum and maximum $\Delta\phi_{lep,anti-lep}$ separation, where $\Delta\phi$ is the angle, in the transverse plane, separating the leptons. The range of $\Delta\phi_{lep,anti-lep}$ is [0,π]. (For W production, this separation is between the charged lepton and the missing energy, neutrino, i.e. $\Delta\phi_{lep,E_{T,miss}}$)

Z **rapidity cutoff for CS frame** - parameter which helps to further define (if the user prefers) the quark direction (assuming a quark/antiquark initial state) in pp colliders. The quark is taken to be moving in the direction of the resulting Z rapidity, which correlates with the true direction. However, the user may choose to employ a minimum Z rapidity cutoff to strengthen this correlation. Collins-Soper

frame related angles ($\cos\theta_{CS}$, ϕ_{CS}) and moments ($A_{0..4}$) are defined with respect to this quark direction, not the generator-level truth.

PDF set - user chosen PDF to implement in calculation: **Note -** see the following *Parton Distribution Functions* section for complete list of sets and other details.

Histogram File

The histogram input file contains the set of predefined histograms, and their respective parameters, that are filled during the running of `FEWZ`. The details of the file follow below.

`Name (DO NOT CHANGE)` - is not an histogram input parameter, and **does not** set the histogram title. This is simply a reference to inform the user which histogram is being set.

`Num of Bins` - sets the number of bins in the histogram. The maximum number of bins per histogram is 25.

`Lower Bound` - sets the lower edge of the histogram.

`Upper Bound` - sets the upper edge of the histogram.

`Write Out Histogram` - parameter setting *which type* and *if* the histogram should be written to file. The *normal* histogram bins the differential cross section in the normal way. The *cumulative* histogram bins the differential cross section, but adds each previous bin to the current one.

- `1` = Write out normal histogram.
- `2` = Write out cumulative histogram.
- `3` = Write out *both* normal and cumulative histogram.
- `0` = Do not write any type of histogram to file.

Definitions of individual histograms are given below.

1-2) Z rapidity and p_T.

3) lepton pair invariant mass.

4-11) lepton and jet pseudorapidity and p_T.

12-13) leading jet pseudorapidity and p_T.

14) beam thrust, defined as

$$\tau_B = \frac{1}{Q}\sum_k |\vec{p}_{k,T}| e^{-|\eta_k - Y|}$$

where Q and Y are the dilepton invariant mass and rapidity, and $\vec{p}_{k,T}$ and η_k refer to the transverse momentum and pseudorapidity of the k-th jet.

15-20) $\Delta R_{ij,separation}$, where $i, j = l, \bar{l}, j_1, j_2$.

21) scalar sum of final state particle transverse momenta, $H_T = \sum_i p_T$, where $i = l, \bar{l}, j_1, j_2$.

22) transverse mass, defined as

$$M_T = \sqrt{2 p_{T,lep} E_{T,miss} (1 - \cos(\Delta\phi_{lep,miss}))},$$

where $p_{T,lep}$ and $E_{T,miss}$ are the transverse momentum of the charged lepton and the missing transverse energy, respectively. $\Delta\phi_{lep,miss}$ is the angle separating the charged lepton and missing energy in the transverse plane.

23) Collins-Soper moments; A_0, A_1, A_2, A_3, A_4 as defined in the angular expansion

$$\begin{aligned}\frac{d\sigma}{dQ_T^2 dY d\Omega_{CS}} = \frac{3}{16\pi}\frac{d\sigma^u}{dQ_T^2 dY}[(1+\cos^2\theta_{CS}) + \frac{1}{2}A_0(1-3\cos^2\theta_{CS}) + A_1 \sin 2\theta_{CS}\cos\phi_{CS} \\ + \frac{1}{2}A_2 \sin^2\theta_{CS}\cos 2\phi_{CS} + A_3 \sin\theta_{CS}\cos\phi_{CS} + A_4\cos\theta_{CS}\end{aligned}$$

where Q_T and Y are the transverse momentum and rapidity of the EW gauge boson, Ω_{CS} are the angles defined in the Collins-Soper frame, and $d\sigma^u$ is the unpolarized differential cross section.

24-25) ϕ_{CS} and $\cos\theta_{CS}$ as defined in the Collins-Soper frame.

26) $\Delta\phi_{ll}$ - angular separation between leptons (lepton/missing energy for W production), measured in the transverse plane.

Smoothing parameters: Fine binning can lead to numerical instability if large opposing weights generated in an evaluation end up in different bins. One option to accelerate numerical convergence is to split the weight of points into two bins, with equal weights in the limit of the boundary. Precisely,

$$w = \frac{1}{2}(1 - (|x - x_i|/y)^\alpha)^\alpha.$$

The bin fraction y is the fraction of the bin near the boundary, x_i, that is subject to smoothing, and α is the level of the smoothing; the weight is guaranteed to vanish or approach 1/2 in the appropriate limit as x^α. w is the fraction of the original weight that gets smoothed to the adjacent bin.

If the user would like to add a histogram to `FEWZ`, there are several files which need to be changed: `include/histos.f`, `src/histogram.f`, `bin/histograms.txt`. **Users change the code at their own risk, and the authors are not responsible for any adverse effects.**

Histograms with varying bin size

As previously mentioned, the user can create histograms with varying bin sizes by calling a separate file, `bins.txt`, from the histogram input file, `histograms.txt`. Rather than specifying the parameters `Num of Bins`, `Lower Bound`, and `Upper Bound` for a particular histogram, the user can insert the name of the file containing the details of the varying bin sizes. Please see the example below.

Consider the histogram for the EW gauge boson transverse momentum. It appears as the following in the histogram input file, `histograms.txt`.

```
'1.  Z pT              ' 20              0d0              660d0            1
```

Here, `'1.  Z pT              '` is the histogram name, `20` is the *number of bins* equally spaced between the *lower bound* of `0d0` and the *upper bound* of `660d0` (in GeV), and `1` tells `FEWZ` to write to file a normal histogram. However, rather than equally spaced bins, the user can choose to have bins of varying size. This is demonstrated in the following line.

```
'1.  Z pT              ' 'bins.txt'         0d0              660d0            1
```

Rather than specify the number of bins, the user simply enters the name of the file, `'bins.txt'` (read by Fortran as a *string*), which details the size of each bin. **Note**: the *lower* and *upper bounds* of `0d0` and `660d0`, respectively, are no longer used, but are now determined by `'bins.txt'`. The following is an example of what `bins.txt` might look like in this scenario.

```
0.0
50.0
100.0
200.0
400.0
```

```
700.0
```

There is no header for this file, only the edges of the bins, separated by new lines.

Each line in the file sets the boundary of a bin. This means that `0.0` is the *lower bound* and `700.0` is the *upper bound*. This file specifies a total of 5 bins:

1) 0.0 – 50.0 GeV

2) 50.0 – 100.0 GeV

3) 100.0 – 200.0 GeV

4) 200.0 – 400.0 GeV

5) 400.0 – 700.0 GeV

If the user wishes to have more than one histogram with varying bins sizes, this is possible. The histogram input file, `histograms.txt`, must be changed appropriately for each histogram with varying bin size. If the desired bins of varying size differ between two histograms, then the bins of varying size must be defined in separate files, i.e. `bins_1.txt` and `bins_2.txt`.

Parton Distribution Functions

PDFs distributed with FEWZ

Below the user can find the complete list of parton distribution functions directly available in `FEWZ`.

ABKM 09: NLO and NNLO sets, called in the input file as `ABKM09NLO` and `ABKM09NNLO`, respectively.

CTEQ: 6L1, 6.5, 6.6, and 10 & 10W sets, called in the input file as `CTEQ6L1`, `CTEQ65`, `CTEQ66`, `CTEQ10`, and `CTEQ10W`, respectively. Grid files for the older sets CTEQ 6.5 and 6.6 are no longer distributed with `FEWZ`, and must be downloaded into your PDF directory.

GJR 08: LO and NLO sets, called in the input file as `GJR08LO` and `GJR08NLO`, respectively.

JR 09: NNLO set, called in the input file as `JR09NNLO`.

MRST 2006: NNLO set, called in the input file as `MRST2006NNLO`.

MSTW2008: LO, NLO, and NNLO sets, called in the input file as `MSTW2008LO`, `MSTW2008NLO`, and `MSTW2008NNLO`, respectfully.

NNPDF2.0: NLO set, called in the input file as `NNPDF20`. The grid files for NNPDF 2.0 are no longer distributed with `FEWZ`.

NNPDF2.1: LO, NLO, and NNLO sets, called in the input file as `NNPDF21lo`, `NNPDF21nlo`, and `NNPDF21nnlo`, respectively.

References for each PDF set can be found in section 3.6 of the paper at
http://arxiv.org/abs/1011.3540 or in the references of the paper at
http://arxiv.org/abs/1201.5896.

Using the LHAPDF library

The user should have a compiled LHA PDF library before installing FEWZ. To compile FEWZ with LHAPDF, the user needs to modify the makefile in FEWZ main directory, changing
LHAPDF = off

to

LHAPDF = on

and specify the LHAPDF library directory using LHADIR variable. The complete list of LHAPDF PDF sets can be found at LHAPDF official website (http://lhapdf.hepforge.org/pdfsets). The user can call each PDF set through specify the grid file name (*.LHgrid) in the input file.

When running with LHAPDF, the PDF directory used by the scripts is ignored. Multisets are automatically detected, but only one may be run at once in this fashion. The scripts will work as normal, but presently are incapable of combining different LHAPDF multisets, which would be required for α_S errors in MSTW, for example. All sets but NNPDF are presently assumed to have asymmetric error eigenvectors, in alternating order.

FEWZ details

This section is dedicated to describing some parts of the code that deserve more attention to better understand the workings of `FEWZ`.

FEWZ makefile

There are several targets of the makefile. They are listed below.

`make`: configures and compiles the Cuba library if not present, then compiles the local-run version of the `FEWZ` executable, and then compiles a Condor-run version of the executable.

`make cuba`: configures and compiles the Cuba library file in the directory `Cuba-1.7`.

`make fewzz`: compiles the Cuba library if not present, and then compiles the local-run version of the `FEWZ` executable for Z production.

`make fewzw`: compiles the Cuba library if not present, and then compiles the local-run version of the `FEWZ` executable for W production.

`make condor_fewzz`: compiles the Cuba library if not present, and the compiles condor-run version of the `FEWZ` executable for Z production.

`make condor_fewzw`: compiles the Cuba library if not present, and the compiles condor-run version of the `FEWZ` executable for W production.

`make clean`: cleans up all the compiled object files.

`make distclean`: cleans up all the compiled object files as well as executables.

Vegas Files

`FEWZ` takes advantage of several Vegas features. After every iteration of the integration, Vegas can write a machine readable file that contains all of the information, including the grid. `FEWZ` takes advantage of this. If, for example, `FEWZ` exits early

(before reaching the desired precision or maximum number of evaluations), due to user intervention or some extenuating circumstance, then the code can be restarted, without any loss in the calculation. Vegas will pick up again where it had left off. This machine-only readable file is named **vegas_last_iter**.

This file is also useful even when a job has finished correctly. Consider a situation where a run of the code exits upon achieving the user defined precision. After running the `finish.sh` script, and examining the final output file, the user decides to run the code to a higher precision. By having the `vegas_last_iter` file, the user doesn't need to start the run from scratch, but simply needs to restart the job by rerunning the `local_run.sh` (or `condor_run.sh`) script with the same arguments. The only needed change is to the Vegas precision setting in the **input** file. Be aware that with Condor, restarting may cause the calculation to be started from scratch when a sector is submitted to a machine with a different architecture than previously.

`FEWZ` also writes a similar file named **entries**. The function of the `entries` file is to store the histogram information and PDF errors. This file is utilized exactly as is `vegas_last_iter`.

Structure of NNLO Calculation

In order to improve the performance of `FEWZ` at NNLO, the differential cross section has been split into pieces called sectors: 133 sectors for Z production and 154 sectors for W production. Each sector is evaluated separately in Vegas, and then merged together when all 133(154) have been evaluated. When running at NNLO, there are 133(154) subdirectories, each containing a copy of the original input file and the resulting output files for that particular sector. This substructure occurs when running locally and with Condor.

Note: The advanced user is advised to pay particular attention to the labeling of the sectors, which can run from 0-132 or 1-133 for Z production and 0-153 or 1-154 for W production. Zero indexing is required for Condor compatibility. However, the user may find a labeling of 1-133(1-154) for various file names. There is a simple mapping from 0-132(0-153) to 1-133(1-154).

FEWZ executable

The local-run version of the `FEWZ` executable can be run outside of the `local_run.sh` script. From the command line, the executable is called as shown below, using Z production as the example.

```
> ./fewzz -i <usr_input_file> -h <usr_histo_file>
              -o <output_file_name> -p <pdf_location> -s <which_sector>
```

The command line arguments have the following definitions.

`<usr_input_file>` is the user-defined input file.

`<usr_histo_file>` is the user-defined histogram input file.

`<output_file_name>` is the name appended for the output files.

`<pdf_location>` is the location of the `dat` directory; if running from the `bin` directory, one can simply use `..`

`<which_sector>` specifies which sector to run and defaults to 0; the sector number specified here is zero-indexed in order to be compatible with Condor.

`-i`, `-h`, `-o`, `-p` and `-s` must be included when calling `fewzz` and its arguments.

The `finish.sh` script can not be used if the local-run version of the `FEWZ` executable is run outside the `local_run.sh` script. `finish.sh` expects the directory structure that `local_run.sh` creates. This is true regardless of the order of perturbation theory that the code was run at. Manipulating the raw output files that are created when running in this fashion requires a more detailed understanding of the scripts contained in the `bin/scripts` directory. Therefore, it is advised that only advanced users of `FEWZ` attempt any running from the command line.

A second executable, `condor_fewzz` or `condor_fewzw`, for Condor system has exactly the same usage. And it is compiled separately to be used by the `condor_run.sh` script. Advanced users can choose to write their own Condor submission file and `condor_fewz` should be the executable to use.

Each executable produces two raw output files for a given sector: a file containing the central value for the cross section and for all histogram bins, and another file containing the same results for all PDF eigenvectors. At this stage the Collins-Soper moments and angles are not yet properly normalized, and PDF errors have not yet been calculated. These operations are performed by the `finish.sh` script as detailed in a later section.

Binary Operations with the Finish Script

`finish.sh` can perform more complicated tasks for multiple runs:

```
> ./finish.sh <run_dir1> <order_prefix>.<output_file1_extension> <operator>
       <run_dir2> <order_prefix>.<output_file2_extension> <new_output_file>
```

where `<operator>` could be

`+`	addition
`-`	subtraction
`*`	multiplication
`/`	ratio
`.`	average
`A`	asymmetry

where the asymmetry operator is defined as

$$A_{ab} = \frac{\sigma_a - \sigma_b}{\sigma_a + \sigma_b}.$$

The average operator will take the weighted average of two runs, such that the numerical error is minimized. This is useful if restarting a calculation is cumbersome or not an option due to differing available architectures.

`<new_output_file>` is the desired new output file name. It takes two independent running directories and performs the specified operation between corresponding integration results (total cross section and histogram bin weight). *The two different runs should have the same histogram configuration for the operation to be performed.* An example would be to obtain the difference between NNLO results and LO results for Z production, using `condor_run.sh` for the NNLO run and `local_run.sh` for the LO run.

```
> ./local_run.sh  z  my_run_LO  my_input_LO.txt
       my_histograms.txt  my_results.dat  ..  1

> ./condor_run.sh  z  my_run_NNLO  my_input_NNLO.txt
       my_histograms.txt  my_results.dat  ..
```

When output files exist in all subdirectories for the NNLO run, the following command can be used:

```
> ./finish.sh my_run_NNLO NNLO.my_results.dat
        - my_run_LO LO.my_results.dat NNLOminusLO_results.dat
```

The file NNLOminusLO_results.dat would contain the data showing the differences of total cross section and histograms of each bin.

How the Run and Finish Scripts Work

Though the FEWZ executable can perform single-sector runs once compiled successfully, it is recommended to use local_run.sh to run FEWZ locally for LO and NLO runs due to the subdirectory structure it creates. For NNLO runs, the structure consists of 133(154) subdirectories inside the run directory specified by <run_dir>, and 133(154) jobs, corresponding to 133(154) NNLO integration sectors, can be started independently. For consistency, the similar structure is created for LO and NLO runs as well, but with only one subdirectory.

Once local_run.sh is invoked with appropriate arguments, the shell script calls scripts/create_parallel.py to set up the subdirectory structure. Given n processors specified by <num_processors> in the argument list, local_run.sh starts n sectors simultaneously in the beginning and waits to start another one when one of the started sectors finishes. It is useful only for NNLO runs because LO and NLO runs contain just one sector. We will assume an NNLO calculation below if not specified otherwise. By default, local_run.sh runs until all sectors finish numerical evaluation. However, there are extra arguments that user can choose to evaluate only the desired sectors:

```
> ./local_run.sh <boson> <run_dir> <usr_input_file> <usr_histo_file>
            <output_file_extension> <pdf_location> <num_processors>
            <init_sector> <last_sector> <sector_step>
```

or

```
> ./local_run.sh <boson> <run_dir> <usr_input_file> <usr_histo_file>
            <output_file_extension> <pdf_location> <num_processors>
            <which_sector>
```

where <init_sector>, <last_sector> and <which_sector> runs from 1-133(1-154), and <sector_step> can be any integer. <init_sector>, <last_sector> and <sector_step> specifies the sector group that the user wants to loop through, i.e.:

for sector from <init_sector> to <last_sector> by <sector_step>

do

...

end do

and only relevant subdirectories are created. <sector_step> can be set to a negative value so that the sector loop goes backward in sector number. Take for example Z production. The user can start two local runs on different machines, one loops from 1 to 133

```
> ./local_run.sh ... 1 133 1
```

while the other loops from 133 to 1

```
> ./local_run.sh ... 133 1 -1
```

Note that the first one can be simply started without the extra arguments using

```
> ./local_run.sh ...
```

since local_run.sh by default loops through all sectors.

The second usage mentioned above will start only one sector as given by <which_sector> and is equivalent to the first usage by setting <init_sector> = <last_sector> = <which_sector> and <sector_step> = 1.

condor_run.sh is designed to start FEWZ jobs on the Condor system where all 133/154 sectors can be submitted to run simultaneously on a computing farm. It calls scripts/create_parallel.py to create the subdirectory structure, and scripts/create_condor_job.py to write the Condor submission file. Output files will be generated and updated in each subdirectory. finish.sh can be used to view a preliminary result whenever output files exist in all subdirectories. If some sector terminates abnormally, the following can be used to submit the job for the corrupted sector individually.

```
./condor_run.sh <boson> <run_dir> <usr_input_file> <usr_histo_file>
          <output_file_extension> <pdf_location> <which_sector>
```

condor_run.sh will try to create the subdirectory structure and original submission file again if not present, and modify the original submission file for the bad sector. The new submission file will be used to submit the single sector job.

Due to the same subdirectory structure, finish.sh is invoked in the same manner for both cases. We recommend local_run.sh or condor_run.sh even for LO and NLO runs, because finish.sh can recognize the unique subdirectory structure for

each type of run, and is able to compile the results more efficiently by calling various python scripts. It determines whether to merge sectors by reading the order prefix of the output file, for which `scripts/merge_parallel.py` is called to generate two raw merged output files in the run directory:

`<order_prefix>.<output_file_extension>` : preliminary merged result for central value;

`<order_prefix>.pdf.<output_file_extension>` : preliminary merged result containing all PDF information necessary to calculate PDF errors.

If the user simply wants to process a single run, `finish.sh` will copy the raw output files to the current directory, i.e. the `bin` directory. If the user wants to perform a binary operation involving two runs, the sectors of both runs will be merged if required by the perturbative order. `finish.sh` will then call `scripts/combine.py`, which produces combined central-value and PDF-error output files in the current directory. At last, `scripts/get_momentA.py` is used to normalize the Collin-Soper angles and moments for both files. PDF errors are calculated by calling `scripts/do_pdf.py`. The extra PDF output file will be deleted, leaving only one final output file. The two raw merged files in the case of `NNLO` runs will be left in the run directory for bookkeeping purposes only.

Note: All shell scripts provide a usage message if called with fewer than the specified number of arguments.